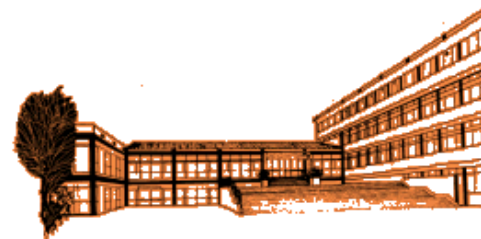




INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ISEL



Instituto Superior de Engenharia de Lisboa

∞
Área Departamental de Engenharia da Electrónica
das Telecomunicações e dos Computadores
∞

Infraestruturas Computacionais Distribuídas

Introspeção



Analisar objectos através da introspeção

Introspeção: Técnica para descobrir e manipular informação sobre objetos ou classes em tempo de execução (*runtime*)

Permite:

- Mostrar informação sobre um objeto
- Criar instâncias de classes cujo nome só é conhecido em tempo de execução
- Invocação de métodos sabendo apenas o seu nome





Introspecção em Java

- Introspecção é suportada pela classe `java.lang.Class`
- Qualquer tipo em Java é representado por uma instancia da classe `java.lang.Class`
- Todos os objetos em Java herdam da classe `java.lang.Object` o método `getClass()`
- Com o objeto da classe *Class* podemos:
 - Perguntar informação acerca de uma classe
 - Descobrir os seus atributos ou métodos
 - Criar novas instancias (objetos) dessa classe
 - Descobrir a super classe, subclasse e as interfaces dessa classe



Métodos úteis da classe *Class* (1)

- Obter o objeto da classe *Class* que representa a classe especificada pelo seu nome

- `public static Class.forName(String className)`

- Retornar o nome completo do objeto da classe *Class*

- `Public String getName()`

- Ex: “java.util.Vector”

- Obter um conjunto de *flags* com a informação acerca da classe, como por ex. se é abstrata, se é uma interface, etc...

- `public int getModifiers()`

- Retornar uma nova instancia do tipo representado pelo objeto desta classe

- `Public Object newInstance()`

- Assume um construtor sem argumentos



Métodos úteis da classe *Class* (2)

 Obter um *array* com todas as classes internas da classe

```
 public Class[] getClasses()
```

 Saber quais os construtores, campos e métodos que existem na classe

```
 public Constructor getConstructor(Class[] params)
```

```
 public Constructor[] getConstructors()
```

```
 public Field getField(String name)
```

```
 public Field[] getFields()
```

```
 public Method getMethod(String Name, Class[] params)
```

```
 public Method[] getMethods()
```

 Descobrir qual o *package* e a super classe


```
 public Package getPackage()
```

```
 public Class getSuperClass()
```




Programar com a classe *Class* (1)

No método toString:

```
 public String toString() {  
    return "A minha classe: " + getClass().getName();  
}
```

Imprimir os nomes dos métodos da própria classe:

```
 public void imprimeMetodos() {  
    Class c = this.getClass();  
    Method[] m = c.getMethods();  
    for(int i=0; i<m.length; i++)  
        System.out.println("Metodo[" + i + "]: " + m[i]);  
}
```



Programar com a classe *Class* (2)

Output gerado:

```
Metodo[0]: public java.lang.String exintrospeccao.MyClass.toString()
Metodo[1]: public void exintrospeccao.MyClass.imprimeMetodos()
Metodo[2]: public native int java.lang.Object.hashCode()
Metodo[3]: public final native java.lang.Class
           java.lang.Object.getClass()
Metodo[4]: public final void java.lang.Object.wait(long,int) throws
           java.lang.InterruptedException
Metodo[5]: public final void java.lang.Object.wait() throws
           java.lang.InterruptedException
Metodo[6]: public final native void java.lang.Object.wait(long) throws
           java.lang.InterruptedException
Metodo[7]: public boolean java.lang.Object.equals(java.lang.Object)
Metodo[8]: public final native void java.lang.Object.notify()
Metodo[9]: public final native void java.lang.Object.notifyAll()
```



Outras classes relacionadas

 Residem no package *java.lang.reflect*

Field

- ⌚ `public Object get(Object obj)`
- ⌚ `public void set(Object obj, Object value)`

Constructor

- ⌚ `public Object newInstance(Object[] args)`

Method

- ⌚ `public Object invoke(Object obj, Object[] args)`



Obter o objeto *Class*

- Todas as classes tem um objecto *Class* correspondente que se pode obter:
 - Escrevendo o nome da classe seguido por “.class”
 - ⌚ Ex: `Vector.class`
 - Invocando o `getClass()` de uma instancia que herde de *Object* (todas)
 - ⌚ Ex: `Vector.getClass()`
 - Invocando o `Class.forName(className)` passando a *String* com o nome do tipo
 - ⌚ Ex: `Class.forName("java.util.Vector")`
 - Carregar uma classe a partir de um dado ficheiro .class, usando o objecto *ClassLoader*
 - ⌚ Ex:

```
ClassLoader cl = ClassLoader.getSystemClassLoader();
cl.loadClass("myPackage.myClass");
```
- Também se pode saber a classe de um objecto recorrendo à keyword *instanceof*
 - Ex:

```
if(x instanceof Circle)
((Circle)x).setRadius(5);
```



Exemplo

```
public class ExIntrospeccao {  
    public static void main(String[] args) {  
        try{  
            Class cl = Class.forName("java.awt.Rectangle");  
            Class[] paramTypes = new Class[] {Integer.TYPE, Integer.TYPE};  
            Constructor ct = cl.getConstructor(paramTypes);  
  
            Object[] constArgs = new Object[] {new Integer(12), new Integer(21)};  
            Object rectang = ct.newInstance(constArgs);  
  
            Method m = cl.getMethod("getWidth", null);  
            Object width = m.invoke(rectang, null);  
  
            System.out.println("O objecto é: " + rectang);  
            System.out.println("A largura é: " + width);  
        }catch(Exception e){e.printStackTrace();}  
    }  
}
```

Resultado:

O objecto é: java.awt.Rectangle[x=0,y=0,width=12,height=21]

A largura é: 12.0



Utilizações da Introspeção

- A introspeção é usada quando se quer examinar ou modificar o comportamento em tempo de execução
 - Facilita a manipulação de classes e objetos referindo nomes
 - Num IDE (*Integrated development Environment*)
 - ⊗ Permite enumerar os membros das classes
 - ⊗ Na execução passo a passo é necessário examinar membros privados
 - Exemplos de uso: JDBC, JavaMail e Jini
- Esta técnica avançada deve ser usada apenas por programadores com plena compreensão dos fundamentos da linguagem Java
- Com essa ressalva, a introspeção é uma técnica poderosa que permite realizar operações que seriam impossíveis
 - Carregar e usar classes e objetos recebidos através da rede



Benefícios

- A introspeção ajuda a manter o *software* robusto
- Pode ajudar a tornar as aplicações
 - Flexíveis
 - Expansíveis
- Pode reduzir o tamanho da imagem em memória (*footprint*) de uma aplicação
- Melhora a performance pois, inicialmente, carrega menos classes
- Carregamento das classes apenas quando são necessárias
- Melhora a reutilização de código
- Permite reduzir código condicional



Desvantagens

- ❏ A introspeção é poderosa, mas não deve ser usada indiscriminadamente, só deve ser usada se não houver alternativa
- ❏ Como a introspeção resolve tipos em tempo de execução, certas otimizações das máquinas virtuais Java não podem ser realizadas, consequentemente, as operações com introspeção têm pior desempenho
- ❏ A introspeção requer privilégios de tempo de execução que podem não estar presentes no contexto de um gestor de segurança
- ❏ Uma vez que a introspeção permite acesso a operações que não respeitam as boas práticas, tais como aceder a atributos e métodos privados, podem ocorrer efeitos colaterais inesperados, que podem tornar o código disfuncional e afectar a portabilidade



“*Bibliografia*” utilizada

 Java tutorial da Introspeção:

 <http://java.sun.com/docs/books/tutorial/reflect/>

 The Java™ Programming Language, 3rd Edition

 Arnold, Gosling, Holmes

 Sun® Microsystems

